USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Consistency

CS 272 Software Development

Professor Sophie Engle
Department of Computer Science

Motivation

# Thread 1: **x++;**	Thread 2: **x--;**
read value of **x**	read value of **x**
calculate **x + 1**	calculate **x - 1**
assign **x** to calculated result	assign **x** to calculated result

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2	calculate **1 + 1 = 2**	
3	assign **x = 2**	
4		read **x = 2**
5		calculate **2 – 1 = 1**
6		assign **x = 1**
7		

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2	calculate **1 + 1 = 2**	
3	assign **x = 2**	
4		read **x = 2**
5		calculate **2 – 1 = 1**
6		assign **x = 1**
7	final value **x = 1**	

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1		read **x = 1**
2		calculate **1 - 1 = 0**
3		assign **x = 0**
4	read **x = 0**	
5	calculate **0 + 1 = 1**	
6	assign **x = 1**	
7		

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1		read **x = 1**
2		calculate **1 - 1 = 0**
3		assign **x = 0**
4	read **x = 0**	
5	calculate **0 + 1 = 1**	
6	assign **x = 1**	
7	final value **x = 1**	

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2		read **x = 1**
3	calculate **1 + 1 = 2**	
4		calculate **1 - 1 = 0**
5	assign **x = 2**	
6		assign **x = 0**
7		

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2		read **x = 1**
3	calculate **1 + 1 = 2**	
4		calculate **1 - 1 = 0**
5	assign **x = 2**	
6		assign **x = 0**
7	final value **x = 0**	

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2		read **x = 1**
3	calculate **1 + 1 = 2**	
4		calculate **1 - 1 = 0**
5		assign **x = 0**
6	assign **x = 2**	
7	final value **x = 2**	

Problems

- Concurrent operations causes **inconsistent** results
- Data shared by threads not **thread safe** access
  - Value may be modified in between read and use
  - Further complicated by caching of values in memory
- Operators x++ and x-- are not **atomic** operations
  - Operations can be divided or interrupted

Thread Safety

- An object is **thread safe** if it maintains a valid or consistent state even when accessed concurrently
- Includes all constants and **immutable** objects
  - `String` or primitive types that are `final`
- Includes some **mutable** objects
  - `StringBuffer`, `java.util.concurrent.*`

Providing Consistency

- If **multithreading**...
  - If **sharing data** between threads...
    - If shared data not already **thread safe**...
      - must **synchronize** access to that data

Synchronization

- Using the **`synchronized`** keyword and intrinsic (or monitor) lock objects to protect blocks of code
- Using the **`volatile`** keyword to protect* variables
- Using **`wait()`** and **`notifyAll()`** to coordinate threads
- Using **conditional synchronization** via lock objects

Synchronization Issues

- Too little synchronization causes **inconsistent** results
  - Code no longer functional
- Too much synchronization causes **blocking**
  - Instead of faster code, results in slower code *(threads can't run concurrently, more complex code)*
  - Can actually cause deadlock*

CHANGE THE WORLD FROM HERE

Software Development
Department of Computer Science

Professor Sophie Engle
sjengle.cs.usfca.edu